UNIVERZITA HRADEC KRÁLOVÉ
UNIVERSITY OF HRADEC KRÁLOVÉ

FAKULTA INFORMATIKY A MANAGEMENTU
KATEDRA INFORMATIKY A KVANTITATIVNÍCH METOD

FACULTY OF INFORMATICS AND MANAGEMENT
DEPARTMENT OF INFORMATICS AND QUANTITATIVE METHODS

MODERNÍ JAVASCRIPTOVÉ FRAMEWORKY
MODERN JAVASCRIPT FRAMEWORKS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MICHAL DOBROVOLNÝ
AUTHOR

VEDOUCÍ PRÁCE — doc. Mgr. TOMÁŠ KOZEL, Ph.D
SUPERVISOR

Hradec Králové 2017

PROHLÁŠENÍ

Prohlašuji, že jsem bakalářskou/diplomovou práci zpracoval/zpracovala samostatně a s použitím uvedené literatury.

Hradec Králové ....

podpis autora(-ky)

PODĚKOVÁNÍ

Toto poděkování bych chtěl věnovat vedoucímu práce, panu doc. Mgr. Tomáši Kozlovi, Ph.D za jeho vedení, cenné rady a podnětné návrhy k této práci.

ANOTACE

Práce se zabývá porovnáním JavaScriptových frameworků pro programování frontend aplikace. Výstupem je ukázková aplikace ve vybraném frameworku - Aurelia, který byl zvolen na základě porovnání jako nejvhodnější volba. K aplikaci byl také vytvořen backend ve frameworku Koa, databáze je realizována pomocí nejpoužívanějšího systému ve spojení s JavaScriptem - MongoDB. Práce dále vysvětluje, jaké má framework Aurelia přednosti a nevýhody. Celá aplikace je navrhnuta dle konvencí frameworku, které zajišťují jednodušší a časově méně náročné rozšiřování aplikace, implementaci úprav a nových částí než v systémech bez využití frameworku a konvencí.

ANNOTATION

The main aim of this work is comparison of JavaScript frontend frameworks. The output is an application implemented in chosen framework - Aurelia. This framework has interesting patterns. Part of this application is also backend which is designed in framework Koa. For database engine was used MongoDB - popular document database. In this work is also explained what are Aurelia's best and worst parts. Whole application is designed using Aurelia's patterns. With this design next expansion and updates will be simpler and less time consuming than in applications without framework and patterns.

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM UKÁZEK

SEZNAM TABULEK

ÚVOD

Tato práce se věnuje oblasti moderního JavaScriptu. Je zaměřena na porovnání frameworků pro tvorbu klientské části. Jejím cílem je zjistit a porovnat, který framework má nejstrmější křivku učení, a zhodnotit kvalitu dokumentací. Hlavním kritériem je pak rychlost vývoje (rapid development).

V současnosti rapidně roste počet JavaScriptových frameworků. Každý z těchto nástrojů jde vlastní cestou, přináší vlastní řešení implementace. Z tohoto důvodu je těžké zvolit správný nástroj pro implementaci konkrétní aplikace. Práce je zaměřena na vybranou množinu Model View Controller (MVC) frameworků, která na rozdíl od full-stack frameworků jsou zaměřeny pouze na klientskou část. Backend aplikace již není řešen tímto nástrojem.

Hlavní přínos této práce spočívá v porovnání frameworků na praktických ukázkách. Ukázky byly navrženy tak, aby demonstrovaly hlavní rozdíly konceptů těchto frameworků. Jelikož se jedná o frameworky postavené na MVC modelu, největší část je věnována porovnání šablonovacích systémů. Pro demonstraci použití zvoleného frameworku Aurelia byla navrhnuta a realizována ukázková aplikace pro správu a vedení úkolů. Aplikace má možnost přihlášení uživatele, správu uživatelů, jejich přiřazení k projektu a možnost komentářů u úkolů.

Práce je strukturována následovně. První část se věnuje stručnému popisu historie JavaScriptu, kdo jej vytvořil, jaký byl jeho účel a čeho se podařilo dosáhnout. Popsán je i jeho vývoj až do současné podoby dle specifikace ECMAScript. Stručně jsou charakterizovány i základní vývojové nástroje, které jsou v současné době téměř standardem. Druhá část je věnována jádru práce - porovnání frameworků. Tato část se zabývá porovnáním komunit kolem frameworků. Porovnávány byly frameworky Angular 2, Aurelia, Backbone a Ember. Třetí část obsahuje popis implementace aplikace ve vybraném frameworku. Ukázkovou aplikací je zjednodušená forma Customer Relationship Management (CRM).

1 JAVASCRIPT A JEHO HISTORIE

Javascript patří mezi interpretované jazyky s možností využití principů objektově orientované programování (OOP). Jeho syntaxe byla převzata z jazyka Java, který sloužil jako inspirace. JavaScript však nemá se samotným jazykem Java nic společného. Nelze jej popsat ani jako jazyk skriptovací, jak by někdo nezainteresovaný z názvu mohl odvodit.

V současné programátorské komunitě je tento jazyk vnímán kontroverzně. Jeho rozdílná implementace v prohlížečích, stejně tak jeho vývoj v posledních letech a vysoké požadavky na znalosti pro vstup do ekosystému, mohou být problémem pro nejednoho programátora.

1.1 Vznik

JavaScript vznikl za účelem vyšší interaktivnosti webového prostředí. Jeho přínos spočíval hlavně v podobě blikajících prvků. Světlo světa spatřil poprvé v roce 1995. [27] Jeho autorem byl Brendan Eich pracující pro Netscape communication. Pro návrh jazyka se inspiroval Javou, Scheme a Self. [13] Cílem bylo vytvořit jazyk podobný jazyku Java, což se také do určité míry povedlo.

JavaScript může být často spojován s jazykem Java, který byl jeho inspirací. Je možno se dopracovat k názoru, že JavaScript je zjednodušenou verzí Javy. Tyto dva jazyky však spolu nesouvisí. Pokud nebude brána v potaz syntaktická podobnost a vlastnost přinést do webové stránky aktivní části (u Javy již nepoužívané), nemají jazyky nic společného. [13]

1.2 ECMAScript

ECMAScript je normalizovaný skriptovací jazyk vedený pod společností ECMA od roku 1997. Tento jazyk má pak své interpretace jako JavaScript, JScript nebo ActionScript. [8] JavaScript je originálním jazykem vytvořeným společností Netscape. JScript je implementací dialektu jazyka JavaScript společností Microsoft. ActionScript je adaptací JavaScriptu využívanéhp platformou flash.

Specifikace ECMAScript jsou vydávány vždy v červnu roku, pro nějž jsou určeny. Tedy ECMAScript 2015 (ES2015) byl vydán v červnu roku 2015.

1.2.1 ES5

Nejpodporovanější verzí ECMAScriptu je ECMAScript 5th edition (ES5), tedy pátá edice standardizovaného EcmaScriptu vydaného v roce 2009. Implementován byl do

všech webových prohlížečů. Po dobu šesti let neproběhla žádná změna ve vývoji. V současnosti je základem JavaScriptu, který podporuje většina zařízení.

1.2.2 ES6 / ES2015

V roce 2015 byla zavedena nová konvence pojmenovávání. Z ECMAScript 6th edition (ES6) se stal z důvodu jednodušší identifikace ES2015. V komunitě se však používá název ES6 velice často.

Do jazyka byly přidány konstanty, block-scope proměnné, block-scope funkce a další. Nejzajímavější novinkou jsou však tzv. arrow funkce [9] a třídy. Arrow funkce umožňují velice krátký zápis funkce. Příkladem může být procházení pole pomocí `[1, 2, 3].map(item => item*10)`.

1.2.3 ES2016

ECMAScript 2016 (ES2016) je specifikací pro rok 2016, jak už nová zkratka napovídá. Nejde zdaleka o tak rozsáhlý update jako v podobě ES2015, ale jde o update zaručující ECMAScriptu posun po malých krocích.

Mezi novinky patří rozšíření integrovaného pole o funkci `[].includes()` a možnost exponenciálního výpočtu pomocí krátkého zápisu dvou hvězdiček (základ ** exponent). [8]

1.2.4 ES2017

Nejnovější specifikace ECMAScript 2017 (ES2017) zatím nebyla oficiálně vydána. Její podoba se stále utváří. Dosud nejzajímavější novinkou jsou pak direktivy `async`/`await`. Ty přidávají velice elegantní možnost tvorby synchronních funkcí pomocí, kdy se `await` použije na místě, kde se má vyhodnotit promise, ve funkci uvedené jako `await`.

1.3 Možnosti využití

JavaScript již není pouze nástrojem k rozblikání stránky. V dnešní době je možno využít jej pro téměř všechny platformy. Nejrozšířenější jsou implementace v aplikacích nazývaných single-page. K rozšíření JavaScriptu došlo také na desktopy, kde pomocí obalovacích frameworků je možno tvořit plnohodnotné desktopové aplikace. Pozadu nezůstalo ani odvětví mobilních aplikací.

1.3.1 JavaScript v prohlížeči

JavaScript je jedním z prostředků umožňující tvorbu interaktivních webových stránek. Jeho hlavním účelem je přidání aktivních prvků, které nefungují pouze na bázi dotaz/odpověď jako např. u PHP. Naopak umožňují implementaci více dynamických prvků a tím tvorbu vysoce interaktivních stránek.

Velkou výhodou využití JavaScriptu v prohlížeči, a tedy i u zmíněných single-page stránek, je vlastnost být serverově nezávislý. Každý návštěvník webu je výpočetní jednotkou, která potřebuje od serveru pouze data. Ty ve většině případů získává pomocí Application Programming Interface (API). Interpretaci už vyřeší sama aplikace běžící na klientské části - v prohlížeči uživatele.

1.3.2 Server

Node.js byl přestastaven na konferenci JSConf 2009 mladým programátorem, který se jmenuje Rayn Dahl. Projekt byl platformou kombinující V8 engine od společnosti Google a událostní smyčky (známé jako event loop), umožňuje použití JavaScriptové aplikace jako serveru. [22]

Tento projekt však neuspokojoval komunitu dostatečně. Kvůli složitosti prosazování změn a rychlosti implementace nových specifikací dle ECMAScript se část komunity rozhodla o odtržení a založila z projektu Node.JS další server-side nástroj s názvem io.js. Vznikem komunitou řízeného nástroje dokázalo io.js v několika prvních měsících přilákat více aktivních vývojářů než Node.js za celou dobu života předtím. [19]

1.3.3 Desktop

Výhodou využití JavaScriptu pro desktopovou aplikaci je jeho podpora všemi platformami. Ve spojení s HyperText Markup Language (HTML) a Cascading Style Sheets (CSS) pak tvoří mocný stroj pro aplikace všeho druhu.

Pro tvorbu desktopové aplikace lze JavaScriptu využít stejně jako například jazyku Java. To díky frameworkům jako Electron nebo Appjs. Electron je v současnosti nejpopulárnějším frameworkem pro tvorbu desktopové aplikace. Jeho hlavní přednosti jsou jednoduchost, podpora nativních API systémů a možnost rychlé tvorby windows instalátoru. [15]

1.3.4 Mobilní zařízení

Aplikace mobilních telefonů se těší v posledních letech veliké popularitě. JavaScript tedy nezůstává pozadu a nabízí několik frameworků pro vývoj na této platformě.

Hlavní motivací pro použití JavaScriptu je fakt, že se jedná o jazyk známý velkému množství vývojářů. Velkým pozitivem je možnost sdílení zdrojového kódu mezi platformami.

Jednou z možností je využít responsivní webovou stránku, kdy není potřeba programovat celou aplikaci pro mobilní telefon. To však není příliš populární řešení z důvodu user experience (UX), výkonu a potřeby být neustále připojen k internetu.

Veliké popularitě se tak těší React Native od Facebooku. Ten využívá nativních komponent prostředí, do kterého je přenesen a je možno jej využít i s kombinací nativních programovacích jazyků. Facebook ho sám aktivně využívá u aplikací jako Instagram, Facebook, Facebook ADS manager. Velké popularitě se těší aplikace Airbnb naprogramovaná v tomto nástroji. [11]

1.4 Základní nástroje vývoje

Vývoj aplikace je v JavaScriptu obehnán velkým množstvím podpůrných nástrojů pro komfort vývojáře. Další motivací k využití těchto nástrojů je zrychlení procesu vývoje. K vývoji moderní aplikace v jazyce JavaScript je často využíváno externích balíčků či celých frameworků. O jejich správu se stará package manager. Dále je třeba kvůli starším prohlížečům a celému ekosystému internetu držet se starších zápisů dle ES5. Programátor však chce využívat nejnovější konstrukce jazyka dle ES2017. Je tedy nutno zajistit transpilaci. O tu se stará webpack. Aby se transpilace spustila při každém uložení zdrojových souborů, je využito gruntu nebo gulpu.

1.4.1 Package manager

Package manager je základním nástrojem pro přidávání závislostí v každé moderní aplikaci. Je znám a dostupný snad ve všech jazycích. Například v PHP je to Composer, v Javě Maven nebo v Ruby RubyGems.

Bez package managera by každý musel vkládat zdrojové kódy utilit, knihoven či frameworků do své aplikace ručně. To by vedlo k mnohem složitější správě závislostí a také vzhledem k chybovosti lidského fakturu i ke zvýšení chybovosti aplikace.

Další z výhod package managera je posloupná správa závislostí. Pokud aplikace závisí na balíčku, který má své vlastní závislosti, package manager je doplní a stáhne také.

NPM je nejrozšířenějším v jazyce JavaScript. Ještě nedávno byl NPM využíván hlavně k instalaci balíčků podporujících funkčnost aplikace. Nyní se jeho působnost rozšířila i na front-end. V současné době jsou z NPM dostupné všechny balíčky, frameworky a utility. Nahradil nástroje, jako je Bower, které byly dříve používány.

Yarn je v současné době nejpopulárnějším package managerem za kterým stojí společnost Facebook. Na githubu se pyšní téměř 22 tisíci hvězdiček a jeho hlavní přednosti jsou bezpečnost, rychlost a spolehlivost. [12] Na rozdíl od NPM vkládá do apliakce Yarn tzv. lock soubor verzí. To znamená, že přesně ty stejné verze, jako jsou nainstalovány nyní, a to i včetně setinových subvezí, budou nainstalovány při každém spuštění příkazu `install`. To je oproti NPM obrovská výhoda. NPM nepodporuje lock soubor a číslo verze instaluje přesně podle specifikace v `package.json`.

1.4.2 Transpilátor

Webpack je tzv. module bundler, tedy nástroj transformující aplikaci do podoby vhodné pro prohlížeč koncového uživatele.

Jeho hlavním přínosem je možnost psát aplikace dle nejnovějších standardů a využívat nejnovějších vlastností JavaScriptu. Příkladem mohou být specifikace ES2015 a ES2016. Tyto vlastnosti nejsou totiž podporovány všemi prohlížeči a Webpack tak umožňuje vývojáři využívat jejich komfortu a přesto tvořit aplikaci kompatibilní i se staršími prohlížeči.

Hot-reload je jednou z dalších výborných vlastností tohoto nástroje. Jedná se o možnost vidět změnu v aplikaci i bez přenačtení stránky, tedy ztráty současného stavu aplikace (state). Možnost je dostupná pouze u vývojového serveru.

1.4.3 Spouštěč úkolů

Spouštěč úkolů je nejčastěji využíván pro kompilace. Příkladem může být kompilace Syntactically Awesome Style Sheets (SASS) na CSS. Avšak jeho využití je mnohem širší. Může se starat o automatické spouštění testů při vývoji, restart dev serveru, kompilaci TypeScriptu na JavaScript nebo kontrolu zdrojového kódu pomocí JSLint.

Grunt je jedním ze dvou nejpopulárnějších spouštěčů úkolů. Grunt se konfiguruje pomocí gruntfile. Ty se pak spouští z client line interface (CLI).

```
1  module.exports = function(grunt) {
2
3    grunt.initConfig({
4      jshint: {
5        files: ['Gruntfile.js', 'src/**/*.js', 'test/**/*.js'],
6        options: {
7          globals: {
8            jQuery: true
9          }
10       }
11     },
12     watch: {
```

```
13        files: ['<%= jshint.files %>'],
14        tasks: ['jshint']
15      }
16    });
17
18    grunt.loadNpmTasks('grunt-contrib-jshint');
19    grunt.loadNpmTasks('grunt-contrib-watch');
20
21    grunt.registerTask('default', ['jshint']);
22
23  };
```

Ukázka 1.1: gruntfile [17]

Dalším spouštěčem je pak Gulp. Zásadní odlišností Gulpu od Gruntu je preferování programování nad konfigurací. Gulp má také výhodu využití pipe pro asynchronní zpracování a vyžaduje ukládání mezistavů. To jej dělá také oproti gruntu rychlejším. Důležité je definovat gulpfile.js. Ten pak vyžaduje, načítá a spouští ostatní. [18]

```
1  var gulp = require('gulp');
2  var pug = require('gulp-pug');
3  var less = require('gulp-less');
4  var minifyCSS = require('gulp-csso');
5
6  gulp.task('html', function(){
7    return gulp.src('client/templates/*.pug')
8      .pipe(pug())
9      .pipe(gulp.dest('build/html'))
10 });
11
12 gulp.task('css', function(){
13   return gulp.src('client/templates/*.less')
14     .pipe(less())
15     .pipe(minifyCSS())
16     .pipe(gulp.dest('build/css'))
17 });
18
19 gulp.task('default', [ 'html', 'css' ]);
```

Ukázka 1.2: gulpfile.js [18]

2 SROVNÁNÍ FRAMEWORKŮ

Výběr správného frameworku může být zásadní pro budoucí vývoj aplikací. Volba frameworku je důležitá, protože se jedná o nástroj, který má za úkol usnadnit vývoj aplikace. Usnadnit tak, aby vývojář nemusel trávit čas řešením běžných úloh, které se dají generalizovat, ale mohl se soustředit na vývoj ostatní logiky aplikace. V JavaScriptu balíčky a frameworky vznikají a zanikají velice rychle. To zvyšuje důležitost volby.

Frameworků pro programování jednostránkové či izomorfní aplikace je v JavaScriptu víc než jen několik. Ve skutečnosti jich je tolik, že přesný počet je těžko dohledatelný. K porovnání byly tedy vybrány pouze tři v současnosti nejzajímavější frameworky - Angular 2, Aurelia, Ember a Backbone. Další možností by určitě mohl být React od Facebooku. Avšak porovnání je zaměřeno vysloveně na frameworky. React je pouze šablonovací knihovna.

2.1 Seznámení s frameworky

Všechny vybrané frameworky mají otevřené zdrojové kódy, podléhají licenci umožňující komerční využití a snaží se vyřešit problematiku vytváření jednostránkové aplikace. Dále využívají konceptů jako MVC či příbuzných. Všechny mají podporu šablonovacích systémů, událostí a směrování. To umožňuje psát aplikace robustní a přitom jednoduše rozšiřitelné.

Angular 2 dostal označení stabilní verze dne 14.9.2016. Mezi jeho zajímavé vlastnosti patří data binding, dependency injection (DI), easy-to-test a rozšíření HTML dialektu pomocí direktiv.

Aurelia je zajímavá především z důvodů integrace web component, databindingu, testovatelnosti a svého šablonovacího systému. Na Aurelii za zmínku také stojí to, že pochází z dílny jednoho z členů týmu Angular 2.0, a to Roba Eisenberga.

Backbone se stal populárním velice rychle, a to hlavně díky tomu, že se jedná o lehkou architekturu MVC. I Backbone obsahuje eventy, celou modelovou vrstvu, kolekce a směrovací část (router).

Ember obsahuje striktnější přístup k vývoji aplikací. To z něj dělá důležitého hráče. Výhodou je možnost rychlé orientace v neznámém projektu, který je vždy psán podobnou konvencí.

Metrika	Angular 2	Aurelia	Backbone	Ember
Hvězdy na Githubu	20 tisíc	9 tisíc	26 tisíc	17 tisíc
StackOverflow otázek	33 tisíc	2 tisíce	20 tisíc	20 tisíc
GitHub přispěvatelů	395	77	292	638
Otevřených issue	945	31	41	213
Vyřešených issue	7779	478	2264	4864
Pull requestů (celkem)	5250	391	1802	6190

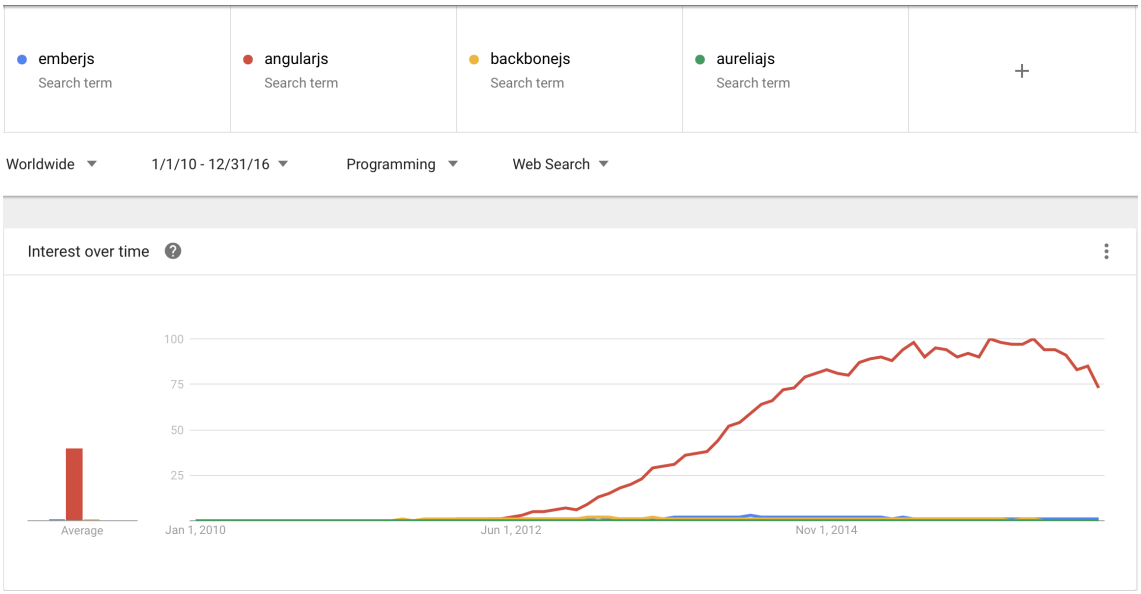
Tab. 2.1: Číselné porovnání komunit

2.2 Komunita

Velká komunita znamená více zodpovězených otázek, více modulů třetích stran, více Youtube tutoriálů, lepší stabilitu a další pozitiva. Proto je komunita jedním z důležitých faktorů ke zvážení při výběru frameworku.

Tabulka 2.1 znázorňuje porovnání čísel z ledna 2017. Data jsou získána z projektových stránek stránek, Github.com, otázek položených na StackOverflow.com. [1] [3] [7] [10]

Dalším faktorem pro volbu mohou také být Google trends. Ty nabízí zobrazení hledanosti jednotlivých výrazů. Tedy nejrychleji rostoucí či umírající frameworky. Na obrázku 2.1 jsou uvedeny trendy vyhledávanosti mezi vybranými frameworky.



Obr. 2.1: Google trends 2010-2016 [16]

2.3 Velikost frameworku

Čas načtení stránky je kritický pro úspěch aplikace. Jakmile dojde na rychlost webové aplikace, koncoví uživatelé nejsou nejtrpělivějším publikem. Tedy je v zájmu všech věnovat pozornost rychlosti načtení.

Velikost frameworku a jeho čas inicializace jsou dva faktory ke zvážení. V obou případech může aplikace ztratit cenné milisekundy. Framework o velikosti více než 500 kB už musí nabídnou něco extra, aby vůbec stál za zvážení. V ideálním případě se mu pak projekt vyhne.

JavaScript bývá za tímto účelem vždy minifikovaný a komprimovaný pomocí gzip. Porovnány jsou tedy verze minifikované a komprimované pomocí gzip komprese verze. Velikost frameworku však může být zavádějící. K jeho chodu jsou totiž často vyžadovány přídavné moduly jako v případě Backbone, kde je to Underscope(5 kB) a jQuery(32 kB) nebo Zepto (9,1 kB).

Metrika	Angular 2	Aurelia	Backbone	Ember
Velikost	766 kB	350 kB	7.6 kB	130 kB

Tab. 2.2: Velikost frameworků

2.4 Šablonovací systém

Jelikož se jedná o část, kterou koncový uživatel uvidí, jde o důležitou součást. Vývojáři či kodérovi by měl usnadňovat reprezentaci dat v HTML. Další výhodou je možnost rozdělit práci mezi role v týmu. Vývojář může odděleně pracovat na logice aplikace a kodér pak psát šablony.

Angular, Aurelia i Ember zahrnují některý z šablonovacích systémů. Vyjimkou je Backbone. Ten nenabízí žádný in-box systém. Nicméně jako výchozí nabízený je uveden Underscore. [6]

2.4.1 Angular 2

Jeho syntaxe je velice jednoduchá a rychle čitelná pro každého, kdo zná HTML.

Pro vypsání hodnoty proměnných se využívá dvou složených závorek. Příkladem může být výpsání názvu frameworku `{{framework.name}}`.

Cykly

Nezbytnou částí života programátora jsou podmínky a cykly. Možností zápisu je hned několik. Nejzajímavější je však zápis s hvězdičkou. Ten z ukázky 2.1. Důvodem

pro využití tohoto způsobu je lepší rozeznání direktiv upravujících strukturu HTML.

```
<ul>
  <li *ngFor="let framework of frameworks">
    {{framework.name}}
  </li>
</ul>
```

Ukázka 2.1: Cyklus nad jednoduchým polem v frameworku Angular 2

Databinding

Šablonovací systém Angularu obsahuje také databinding, neboli provázání proměnné s šablonou. Jeho úkolem je zajistit zobrazení hodnoty proměnné, pokud se změní stav aplikace v závislosti na čase nebo akci. Obousměrný binding pak zajišťuje také možnost načítat hodnotu proměnné například z HTML inputu do aplikace a pak ji zpětně reprezentovat, jak je uvedeno v ukázce 2.2. Ukázka neobsahuje pouze šablonu. Angular vyžaduje v takovém případě kompletnější kód obsahující i komponentu.

```
import { Component } from '@angular/core';

@Component({
  selector: 'hello-angular',
  template: `Fill name: <input type=text [(ngModel)]="name"><br>
  <p>Hello {{name}}!</p>`
})
export class HelloComponent {
  name = "angular 2";
}
```

Ukázka 2.2: Obousměrný databinding v frameworku Angular 2

Komponentu z ukázky 2.2 je pak možno vykreslit pomocí kódu z ukázky 2.3.

```
...
<body>
  <hello-angular></hello-angular>
</body>
...
```

Ukázka 2.3: Vykreslení komponenty v frameworku Angular 2

Události

Další částí jednostránkové aplikace jsou události. Pomocí událostí aplikace reaguje na uživatelovy podněty. Například kliknutí, změnu hodnoty nebo jiné. Angular umožňuje zápis dvěma způsoby. První je pomocí závorek `(click)=`. Další možností

je pak využití prefixu `on-`. Ukázka 2.4 obsahuje oba případy. Kliknutím na tlačítko se pustí akce `onClick()`.

```
1 <button (click)="onClick()">
2 <button on-click="onClick()">
```

Ukázka 2.4: Události v frameworku Angular 2

2.4.2 Aurelia

Zápis šablon ve frameworku Aurelia je člověku znalému HTML bezproblémový. Avšak oproti Angularu 2 je hůře zapamatovatelný. U šablony je vyžadováno, aby byla zabalena značkou `<template></template>` a používala příponu HTML.

Proměnné

Proměnné se v Aurelii zapisují stejně jako Template Literals v ES6 pomocí složených závorek a dolaru před nimi. Příkladem může být zobrazení vlastnosti name z proměnné framework `${framework.name}`.

Cykly

V Aurelii jsou dostupné 2 typy podmínek sufixované výrazem `.bind`. Příkladem je `if.bind="isAvailable"`. První možností podmínky je tedy `if`. Ta při nesplnění vynechá zcela prvek z HTML. Druhou možností je `show`. Ta při vykreslování rozhodne, zda má být prvek doplněn třídou `aurelia-hide`. Třída `aurelia-hide` může být modifikována pomocí vlastního CSS. Výchozí vlastnost je pak `display: none;`.

Cykly Aurelia nabízí typy iterace nad polem, rozsah, iteraci objektu Set, klíč a hodnota (map) či Objektu obecně. V ukázce 2.5 je znázorněna iterace nad jednoduchým polem.

K iteracím je přidáván prefix `repeat` s tečkou.

```
1 <template>
2   <ul>
3     <li repeat.for="framework of frameworks">
4       ${framework.name}
5     </li>
6   </ul>
7 </template>
```

Ukázka 2.5: Cyklus v frameworku Aurelia

Databinding

Databinding se v Aurelii specifikuje sufixem `.bind`. Navazovat pak lze veškeré proměnné přidružené komponenty. V ukázce 2.6 je znázorněn příklad zdravící komponenty reagující na vstup uživatele.

```
1 <template>
2   Fill name: <input type=text value.bind="name"><br>
3   <p>Hello ${name}!</p>
4 </template>
```

Ukázka 2.6: Šablona využívající databinding v frameworku Aurelia

Na rozdíl od Angularu 2 (ukázka 2.2), Aurelia nezapisuje komponentu (ukázka 2.7) a šablonu do jednoho souboru. Je tedy třeba ještě vytvořit jeden další JavaScriptový soubor obsahující proměnné využívané v šabloně.

```
1 export class Greeter {
2   constructor() {
3     this.name = 'Aurelia';
4   }
5 }
```

Ukázka 2.7: Komponenta v frameworku Aurelia

Události

Aurelia zapisuje události podobným způsobem jako binduje data. K provázání je využito sufixu `.bind`. Například událost kliknutí je pak demonstrována ukázkou 2.8.

```
1 <button click.bind="onClick">
```

Ukázka 2.8: Událost kliknutí v frameworku Aurelia

2.4.3 Backbone

Jako jediný framework z výběru, Backbone nenabízí přiložený šablonovací systém v balíčku. Lze ho velice jednoduše integrovat s různými knihovnami třetích stran. Na oficiálních stránkách je však doporučeno použít Underscore.

V Undescore se vše zapisuje v podobě JavaScriptu zapsaného mezi `<% ... %>`. Konkrétní funkce se pak volají nad podtržítkem. Underscore je stejně jako Backbone minimalistický nástroj. Undescore není přímou součástí Backbone. Nebude mu tedy věnována taková pozornost.

Proměnné

Undescore vypisuje proměnné automaticky uvedením mezi šipku, procento a rovnítko `<%= ... %>`. Pokud je žádoucí výpis HTML, použije se místo rovnítka (=) pomlčka (-).

Cyklus uvedený v příkladu 2.9 vypisuje frameworky do seznamu za využití Undescore. Je zjevné, že se nejedná o rozsáhlou knihovnu, ale o lehkou nadstavbu nad HTML.

```
1 <ul>
2   <% _.each(frameworks, function(framework) { %>
3     <li>
4       <%= framework.name %>
5     </li>
6   <% }); %>
7 </ul>
```

Ukázka 2.9: Cyklus v frameworku Underscore

2.4.4 Ember

Šablonovací systém frameworku Ember je Handlebars. Ten je vybudován nad populárním Mustache. Jako přípona souboru je použita zkratka `.hbs`.

Proměnné

Pro zobrazení obsahu proměnných se využívá složených závorek `{{framework.name}}`.

Cykly

Iterování v Handlebars je zapisováno trochu výřečnějším způsobem. Před příkaz se zapisuje mřížka a proměnná, která je výstupem každého průchodu, a uvozuje se svislou čarou. Zápis je patrný z ukázky 2.10.

```
1 <ul>
2   {{#each frameworks as |framework|}}
3     <li>
4       {{framework.name}}
5     </li>
6   {{/each}}
7 </ul>
```

Ukázka 2.10: Cyklus v frameworku Handlebars

Handlebars podporuje velice zajímavou možnost zobrazení prázdného pole. Místo podmínky se uvede značka `{{else}}`.

Další zajímavou vlastností je automatické rozbalování proměnných. Pokud není v cyklu uveden název proměnné ve svislých čárách, obsah je automaticky rozbalen na vnitřní proměnné (ukázka 2.11).

```
<ul>
  {{#each frameworks}}
    <li>
      {{name}}
    </li>
  {{else}}
    <li>Ember</li>
  {{/each}}
</ul>
```

Ukázka 2.11: Cyklus for s else větví v frameworku Handlebars

Komponenta

Ember stejně jako Aurelia (ukázky 2.6 a 2.7) rozděluje komponenty na dvě části kódu. První je funkčnost kontroly (ukázka 2.13) a druhou částí je šablona samotná (ukázka 2.12). To je velice užitečné pro vývojáře. Ten má v takovém případě oddělenou logiku od šablony.

```
Fill name: <input type='text' value=name><br>
<p>Hello {{name}}!</p>
```

Ukázka 2.12: Šablona komponenty v frameworku Handlebars

```
import Ember from 'ember';

App = Ember.Application.create();
App.ApplicationController = Ember.Controller.extend({
  name:'Ember',
});
```

Ukázka 2.13: Databinding v frameworku Ember

Události

Ember řeší události pomocí helperu `action` s parametrem názvu akce. Dále nabízí možnost specifikovat, při jaké události myši nebo klávesnice se událost spustí. Jednou z dalších možností je přímo v šabloně definovat, zda se má nad prvkem spustit výchozí akce prohlížeče pomocí `preventDefault=true/false`.

```
<button {{action "click" on="click"}}>
```

Ukázka 2.14: Události v frameworku Ember

2.5 Angular 2

Angular obsahuje mnoho inovativních myšlenek. To je kritické pro framework ve světě vývoje internetových stránek.

Jelikož jednou z hlavních vlastností programátora je lenost, moderní frameworky či knihovny musí poskytovat co nejkratší a nejjednodušší zápisy. Jako příklad lze uvést porovnání s jQuery (ukázka 2.15) kde textový vstup je propisován do těla HTML stránky.

```
1 $('#form input#name').on('value', function() {
2     $('#form div').text('Hello ' + this.val() + '!');
3 });
```

Ukázka 2.15: Živě propisovaný textový vstup v jQuery

Stejnou funkčnost lze v Angularu 2 zapsat velice krátkým způsobem (ukázka 2.16).

```
1 <input [(ngModel)]="name" type="text" />
2 Hello {{name}}!
```

Ukázka 2.16: Živě propisovaný textový vstup v frameworku Angular 2

2.5.1 Pozitiva

Angular má jednou z největších komunit kolem webového frameworku. Projevuje se to i tím, že patronem Angularu je Google.

Výkon

Angular 2 oproti Angularu 1 velice zapracoval na výkonu. Ve srovnání s první verzí se jedná o nesrovnatelný skok.

CLI

Angular 2 nabízí vlastní CLI nástroj. Pomocí příkazu `ng` je pak vývojář schopný velice rychle zakládat nové aplikace, zakládat komponenty, spouštět vývojářský server atd..

Výhodou vývojářského serveru je to, že není třeba instalovat na každý stroj složité nástroje, jako je Apache, a nastavovat virtualhost. Vše je vyřešeno jednoduchým příkazem `ng serve`.

Aplikace je spuštěna a ihned dostupná na adrese místního počítače (localhost) pod nakonfigurovaným portem. Vývojářský server také zajišťuje automatickou obnovu prohlížeče při změně ve zdrojových kódech (hot-replace).

IDE

Důležitou součástí frameworku je podpora Integrated development environment (IDE). Pokud framework nabízí podporu IDE, vývojáři je nabídnut veliký komfort.

Základem podpory IDE jsou útržky kódu (snippet), napovídání (autocomplete) a statická analýza kódu.

Další výhodou Angularu 2 je využití TypeScriptu. Ten je široce podporován editory a nabízí tak i bez speciálního modulu vyšší komfort vývoje.

TypeScript

JavaScript a TypeScript jsou dvě nabízené možnosti zápisu zdrojových kódů v Angularu 2. Tím preferovanějším je TypeScript. Ten je také nabízen jako výchozí možnost v oficiální dokumentaci.

Základní vlastností TypeScriptu je transpilace do JavaScriptu. Výstupem je tedy vždy JavaScript. TypeScript však na rozdíl od JavaScriptu nabízí některé rozšířené možnosti a prvky objektového návrhu.

Dalším zajímavým prvkem Typescriptu je pak typová kontrola. Ta poskytuje vývojáři statickou kontrolu a případné predikce chyb ještě před samotným spuštěním. V kombinaci s kvalitním IDE se může jednat velice cennou úsporu času.

Testování

Angular 2 je od začátku navržen k jednoduchému testování. Pro jednotkové (unit) testy Angular preferuje nástroj Karma. Karma byla dříve známá pod jménem Testacular. Jedná se spouštěč testů s velice širokou podporou ve vývojářských prostředích.

Druhým nástrojem je Protractor. Tento nástroj je využíván pro aplikační testy, jež jsou psány a spouštěny nad aplikací, jako by se jednalo o reálného uživatele používajícího aplikaci.

Animace

Jedním z integrovaných modulů jsou animace. Nabízeno je rozhraní, pomocí kterého lze zapsat animace podporující UX změn stavů aplikace.

Příkladem může být tlačítko aktivní / neaktivní měnící stav uživatele. Při kliknutí je tlačítku změněna třída a Angular 2 se postará o zbytek. Uživatel je veden pomocí grafické animace změnou stavu.

```
1 animations: [
2   trigger('state', [
3     state('inactive', style({
4       backgroundColor: '#eee',
5       transform: 'scale(1)'
```

```
6       })),
7       state('active',   style({
8         backgroundColor: '#cfd8dc',
9         transform: 'scale(1.1)'
10      })),
11      transition('inactive => active', animate('100ms ease-in')),
12      transition('active => inactive', animate('100ms ease-out'))
13    ])
14  ]
```

Ukázka 2.17: Animace tlačítka v frameworku Angular 2

2.5.2 Negativa

Negativ má Angular několik, ty nejzajímavější budou následně popsána. Příkladem může být také velikost, 766 kB (tabulka 3.1) je opravdu nezanedbatelné číslo. Angular si prošel bouřlivým vývojem. V době, kdy se již mělo jednat o Release Candidate (RC), byly stále prováděny Breaking-change (BC). To není ideální značka pro komerční framework, který ovládá trh.

Šablonovací systém

HTML Angularu 2 není HTML. HTML značky nejsou citlivé na velikost písmen, ale Angular 2 značky jsou. Může se stát, že vývojář direktivu `ngIf=` napíše jako `ngif=` a pak stráví hledáním chyby cenný čas.

Další nevýhodou mohou být různé zápisy direktiv. Jednou je použita hvězdička, podruhé hranaté závorky a nakonec se může stát, že se vše zapíše do kulatých závorek a poté obalí hranatými. To definitivně všechny nováčky zmate.

Také pokud je `*ngIf` použit u značky `<template>`, jedná o direktivu ovlivňující výsledné HTML. Vygenerované HTML bude vypadat takto `<template [ngIf="true"]><template>...</template></template>`

2.6 Aurelia

Nejmenší komunitu má Aurelia. Je také nejmladším z frameworků, ale i přesto je více než konkurenceschopná.

2.6.1 Pozitiva

Mezi pozitiva Aurelie určitě lze zařadit rozmanitost a poctivost. Také vývojářským komfortem se řadí mezi přední nástroje. Její DI, komponenty nebo šablony jsou

velice přehledné a čitelné. Také dokumentace Aurelie je nejpřehlednější z vybraných frameworků. Obsahuje dva návody, jak postavit aplikaci od prázdné složky. Dalším plusem dokumentace je plynulost poskytnutých informací.

Podpora jazyků

ES5, ES2015, ES2016 nebo TypeScript, ve všech vyjmenovaných jazycích je možno psát aplikaci s využitím frameworku Aurelia. Dokumentace je také nabízí jako možnost volby u vložených.

Sestavovací systém

Aurelia podporuje JSPM, Webpack a Aurelia CLI. Při zakládání projektu je vývojář vyzván ke zvolení jednoho z vyjmenovaných. Každý má pak drobně odlišnou konfiguraci či spouštění. Aurelia je jediným z vybraných frameworků, který nabízí oficiální podporu a možnost výběru.

Příkazová řádka

Příkazem `au` lze vyvolat vlastního CLI manažera. Ten v Aurelii nabízí možnosti jako vygenerovat nový projekt nebo jej zkompilovat, dále také spustit v prohlížeči.

Nástroj nabízí různé možnosti dle toho, v jakém kontextu se nachází. V běžné složce nabídne možnost vytvoření nového projektu. Pokud však pracovní adresář již je Aurelia projektem, nabídne možnosti jako spuštění, testování či generování.

Při zakládání nového projektu je vývojář dotazován interaktivním způsobem na vlastnosti projektu. Aurelia CLI pak vytvoří prázdný projekt, který bude možno ihned spustit a bude odpovídat preferencím vývojáře.

Předávání závislostí

Aurelia nabízí velice elegantní způsob jak, předávat závislosti. Je jím metoda, jež se vloží před definici třídy `@inject(...)`. Třídě je pak při instanciaci předáno do konstruktoru vše, co bylo zadáno metodě.

Čisté komponenty

Aurelia je jediným z frameworků umožňujících tvorbu komponent pouze pomocí čistého JavaScriptu. V Angularu 2 to bylo například pomocí dekorátoru `@component`, která se pak provazovala s třídou.

EventAggregator

Aurelia umožňuje zasílání a naslouchání na tzv. zprávy pomocí třídy EventAggregator. Pro jejich použití je třeba definice naslouchání `ea.subscribe('Event', callback(msg))` a zasílání `ea.publish('Event', msg)`. Při odeslání zprávy jsou pak notifikováni všichni posluchači a vyvolána jejich aktivita.

Učení

Aurelia nabízí jednoduché, ale komplexní příklady, které mají za úkol provést vývojáře možnostmi frameworku. Křivka učení je strmá a již po několika minutách je vývojář schopný se orientovat.

Pochopení Aurelie značně pomáhá její jednoduchost a přehlednost. Komponenty psané pouze v JavaScriptu a šablony nevyžadující složité názvy atributů jsou velkým pozitivem.

2.6.2 Negativa

Aurelia stejně jako všechny ostatní má své body ke zlepšení. Komunita Aurelie je nejmenší z vybraných frameworků a také stabilita CLI nástroje mohou být příkladem.

Komunita Malá komunita s sebou přináší některé nevýhody a i když se nejedná o část, kterou může framework příliš ovlivnit, je třeba se jejím rozvojem aktivně zabývat.

Při hledání příkladů se často stává, že nalezené řešení již není kompatibilní s novou verzí, dokumentace nestačí a programátor s nedostatkem zkušeností se tak dostane do problémů.

Dalším příkladem může být nedostatečně pokrytá část frameworku. V takovém případě pak je vývojář odkázán pouze na zdrojový kód.

Při čtení tutoriálů nebo dokumentace také často dochází k nedostatečnému pochopení problematiky. Proto je výhodou velkých komunit množství různých příkladů, které pomohou.

Příkazová řádka V režimu vývoje je standardně projekt servírován prohlížeči pomocí vývojářského serveru. Ten se spouští příkazem `au run`. Pokud je žádoucí ihned po změně ve zdrojových kódech vidět změnu v prohlížeči, přidá se parametr `-watch`. Pokud však vývojář udělá například syntaktickou chybu a soubor uloží, server se pokusí o transpilaci zdrojových kódů a celý spadne. Je tedy nutno se opět vrátit do konzole a znovu jej spustit.

2.6.3 Externí balíčky

Při vývoji aplikace je dnes běžné neprogramovat znovupoužitelné části, ale používat balíčky či knihovny. Aurelia jich nenabízí příliš mnoho. Navíc většina z nich již není kompatibilní nebo využívá zastaralé balíčky, či balíčky samotné Aurelie, které již nejsou podporovány.

HTTP klient

Klientem pro dotazování serveru v Aurelii může být ‘aurelia-http-client‘ nebo ‘aurelia-fetch-client‘. Vývojář tak musí hledat i při tak triviální věci, jako je načítání dat, nápovědu aby se nejdříve rozhodl, který nástroj je vhodnější a pak jej později konkrétně použít.

2.7 Backbone

Backbone je tím nejtenčím řešením z vybraných frameworků. Jeho stopa ve výkonu aplikace je tím pádem mizivá.

2.7.1 Pozitiva

Křivka učení je přímá. Backbone díky své jednoduchosti nabízí rychlou implementaci aplikace pomocí konstrukcí, které nabízí.

Dokumentace

Backbone má stručnou a jednoduchou dokumentaci. Díky tomu je čtení dokumentace rychlé a srozumitelné. Dokumentace také disponuje spoustou příkladů a hotových řešení. V neposlední řadě pak obsahuje ukázku projektů, které Backbone používají.

Celá dokumentace je napsána jako jediná webová stránka.

Jednoduchost

Protože je Backbone tak malý a jednoduchý, nabízí velice mocný nástroj pro tvorbu aplikace. Výsledný nástroj totiž ve výsledku je tvořen několika vývojářem vybranými knihovnami a zásuvných modulů. Výsledný framework si vývojář vytvoří sám nad Backbone. Dostává svobodu volby pro šablonovací systém, router a další. U Angularu 2, Aurelie nebo Emberu je třeba žít s tím, co je již ve frameworku zahrnuto.

2.7.2 Negativa

Backbone neposkytuje strukturu a striktní konvence. Dává tím vývojáři velkou volnost v implementaci. Jedná se spíše o základní nástroj. Všechny složitější problémy musí řešit vývojář.

Jednoduchost

Stejně jako je jednoduchost výhodou Backbone, je i jeho nevýhodou. Vývojář totiž musí projít spoustou balíčků a zásuvných modulů třetích stran, aby si vybral. Rozhodování, který balíček je nejvhodnější pro konkrétní případ, může trvat dlouho.

Backbone podporuje databiding na základní úrovní. Pro změnu zobrazení bude tedy třeba naprogramovat spoustu kódu.

Testování

Jednotkové testy je komplikované psát, protože s DOM dokumentem je v Backbone manipulováno přímo. Stejně tak je přímou manipulací ovlivněna znovupoužitelnost.

2.8 Ember

Ember využívá přísné struktury aplikace. Nutí vývojáře držet se jeho konvencí a postupů. Pak je každá aplikace psaná v Emberu velice podobná a programátor se v ní rychle orientuje.

2.8.1 Pozitiva

V Emberu je zahrnut skvělý router stejně tak jako datová vrstva, zvaná ember data.

Konvence

Konvence jsou upřednostňovány nad konfigurací. To je užitečný koncept nabízející spoustu automaticky generovaného kódu, který by jinak vývojář muset napsat ručně (nevýhoda Backbone).

Příkladem může být Router. Ember je schopný automaticky generovat jméno routy, pokud jej vývojář sám nedefinuje.

Datová vrstva

Ember obsahuje plně vyvinutou datovou vrstvu na rozdíl od Angularu a Backbone. Datová vrstva se integruje jednoduše oproti jakémukoliv JSON API. Požadavek na

API je pouze, aby následovalo několik zásadních konvencí. Ember už pak zajišťuje vše ostatní.

Datová vrstva také nabízí možnost testování. Je připravena na testování jednoduchých objektů i jejich vztahů.

Výkon

Životní cyklus Ember aplikace je nazýván "run loop". Jeho užitečnost spočívá v tom, jak řídí běh aplikace. Postupně jednu po druhé spouští části životního cyklu. Jedním z jeho benefitů je pak možnost vůbec nepřekreslovat znova šablonu, pokud to není nutné (nezměnila se data).

Ember také profituje z předkompilovaných šablon Handlebars. Ty jsou předem připraveny pro rychlé využití při vykreslování.

2.8.2 Negativa

I přestože Ember je velice povedený framework, má několik nevýhod a jeho popularita není nejvyšší.

Komunita Ember není tak rozsáhlá jako například u Angularu. To může vést ke komplikacím s řešením případných problémů vývojáře.

Framework je také značně otevřen příspěvkům od nezávislých vývojářů. To může vést k zavádění chyb nebo nekonzistencí do jádra.

Nestálost

API frameworku se často mění. Nejen ve velkých, ale i v drobných verzích. Tím vzniká na StackOverflow velké množství nevalidního obsahu. Příkladem může být vychvalovaný datový model. V jeho vývoji je tolik BC, že se StackOverflow může stát velice zavádějící pomocí.

Šablonovací systém

Handlebars generuje mnoho značek `<script>` do HTML. Navíc šablonovací systém může díky této vlastnosti snadno rozbít kompatibilitu s ostatními frameworky nebo knihovnami. Příkladem může být jQuery.

3 ANALÝZA A NÁVRH APLIKACE

Aplikace jako seznam úkolů se zatržítkem je příliš jednoduchá na to, aby ukázala nekomfortní zóny frameworku. Je třeba zvolit aplikaci, která je dostatečně komplikovaná na to, aby obsahovala zabezpečenou sekci, několik různých oprávnění uživatelů, entity obsahující vazby a složitější entity obsahující více než jeden vstupní parametr.

3.1 CRM systémy

CRM jsou systémy sloužící ke správě pracovních vztahů mezi zákazníkem a společností. Dalším z úkolů je také sloužit jako datová základna k ukládání historie komunikace nebo také ke správě zakázek. CRM systémy se také často používají k měření výkonnosti společnosti či jednotlivých zaměstnanců. Použitým principem může být například doba, za jakou je požadavek označen jako ukončený od založení. Systém CRM se také často využívá jako nástroj k zaznamenávání odpracovaného času na jednotlivých úkolech nebo projektech. Složitější implementace těchto systémů přináší také ke správě úkolů ekonomické prvky. Jako příklad lze uvést kalkulace dle náročnosti požadavku, vystavování faktur a tvorbu přehledů.

CRM aplikace je s určitou mírou abstrakce aplikace typu úkolníku. Uživatel zadává úkol, který je třeba vyhotovit. Po vyhotovení je označen jako ukončený. CRM přidává do tohoto jednoduchého procesu několik dalších prvků. Těmito prvky jsou uživatelé, uživatelské role a třeba možnost diskusí u úkolů.

3.2 Popis aplikace

Cílem ukázkové aplikace je možnost založit požadavek k řešení. K požadavku je poté možno vést diskusi. Do diskuse může přispívat kterýkoliv uživatel přidělený k projektu. Uživatel může kdykoliv označit požadavek jako vyřešený. Vyřešený požadavek je možno kdykoliv vrátit zpět do stavu nevyřešený.

Základem CRM systému je projekt. Tomu je možno přiřadit uživatele, kteří se na něm podílejí, případně pak zákazníky (stejná položka). Úkolem projektu je agregovat požadavky do jedné skupiny a také zastřešit přístupy uživatelů. Klient má možnost pouze vidět požadavky, které jsou přiřazeny jeho projektům. Registrace klientů může provádět pouze uživatel s oprávněním administrátor.

Požadavky jsou provázeny vlastní časovou osou. Do osy se zaznamenávají komentáře, které uživatelé přidali. Časová osa se zobrazuje v detailu požadavku s informací o autorovi a času u každého komentáře.

3.3 Backend API

JavaScriptová aplikace potřebuje i serverovou část, která pracuje s databází. Ta se nazývá API. Klientská část je sama o sobě a s mírnou nadsázkou pouze zobrazovacím prostředkem dat. Celá aplikace pak funguje jako klientská část dotazující se na serverovou část ohledně dat. Odpovědi ze serveru jsou zpracovány a zobrazeny uživateli v přívětivé podobě. Klientská část žádná data nemá a obsahuje pouze jejich popis v podobě transformací na čitelné informace. Serverová část naopak nemá žádné prvky uživatelského prostředí. Serverová část pouze na určité dotazy vrací datové, případně chybové odpovědi.

Serverová část může být napsána v libovolném jazyce. Důležité je, aby obě části dodržovaly konkrétní strukturu dat. Vzhledem k charakteru bakalářské práce však byl také zvolen JavaScript.

3.4 Uživatelské rozhraní

Uživatelské rozhraní je nejdůležitější součástí aplikace z pohledu uživatele. Uživatel se nezajímá, jak je aplikace napsána nebo jaká technologie za ní stojí. Uživatel ocení hlavně přehledné a jednoduché uživatelské rozhraní. Obor zabývající se touto problematikou se nazývá UX. Ukázková aplikace však není komerční a je použito pouze několik základních pravidel pro tvorbu moderní internetové aplikace.

Uživatelské rozhraní je měněno dle přihlášeného uživatele. Každý uživatel má jiné možnosti přiděleny dle své uživatelské role a využívá systému za jiným účelem.

3.5 Uživatelé a uživatelské role

Uživatelé jsou rozděleni do tří skupin. Uživatelská role určuje, jakou část aplikace uživatel má právo navštívit a spravovat. Případně pak, jaké události jsou uživateli umožněny. Ať už se jedná o manipulaci uživatelů, projektů nebo samotných úkolů.

Role od sebe postupně dědí. Každá vyšší role automaticky zahrnuje práva nižší role. Posloupnost je následovná: klient je níže než vývojář, vývojář je níže než administrátor.

Role	Rozšiřuje práva role	Práva
Uživatel	-	přidat požadavek, přidat komentář
Vývojář	Uživatele	správa požadavků, správa projektů
Administrátor	Vývojář	správa uživatelů

Tab. 3.1: Seznam uživatelských rolí

Klient je nejnižší rolí v hierarchii. Jeho práva jsou pouze přihlášení a zakládání úkolů či událostí, jako jsou komentáře. Úkolem této role je umožnit vnitřním pracovníkům společnosti spravovat úkoly. Příkladem může být změna stavu požadavku na vyřízený. Vývojář má práva přijímat, editovat, či zakládat úkoly a projekty. Jediným omezením této role je správa uživatelů. Vývojář nemá právo zakládat uživatele ani je editovat. Nejvyšším uživatelským oprávněním je administrátor. Ten má jako jediný právo zakládat uživatele. Zároveň zahrnuje všechny možnosti předchozích rolí.

3.5.1 Projekty

Rolí projektu v návrhu aplikace je seskupení úkolů a identifikace, o jakou aplikaci se jedná. Z toho vyplývá, že musí být pro každou jednotlivou zakázku vytvořen jeden projekt v aplikaci. Touto vazbou se také pak jednodušeji přidělují uživatelé k úkolům. Každý uživatel bude mít zobrazeny pouze úkoly, které patří pod jeho kompetenci, a nebude se ztrácet ve velkém množství nerelevantních úkolů.

Projekty jsou jedním ze základních stavebních kamenů aplikace, avšak jedná se o velice jednoduchou entitu. Její vlastnosti jsou pouze název, adresa v internetu, záznam o aktuálnosti a seznam přidělených uživatelů. Nejdůležitějším vztahem je seznam přidělených uživatelů. Jak již bylo zmíněno, přidává uživateli práva pracovat s úkoly přidělenými k projektu. Pokud uživatel není přidělen k projektu, neuvidí požadavky ve svém seznamu a nebude moci přispívat do diskusí.

3.5.2 Úkoly

Úkol je podnět k úpravě, opravě nebo dotaz. Kdykoliv chce klient komunikovat se společností, založí nový úkol. Ke každému úkolu je evidováno, kdo a kdy jej založil. Dále také, k jakému projektu patří, jeho stav a název. Stav slouží k označování požadavku jako nového či hotového. Úkol je možno označit jako vyřešený v jeho detailu. Tato možnost se nachází hned pod časovou osou komentářů.

Komentáře Zanořenou entitou úkolu jsou komentáře. Jsou reprezentovány jako časová osa v detailu úkolu. Vložení komentáře je možno pod časovou osou. Komentáře nejsou omezeny dle stavu úkolu. Je tedy možno komentovat jakýkoliv úkol kdykoliv, nehledě na to, zda je vyřešený nebo nově zadaný.

4 POPIS IMPLEMENTACE

Aplikace se skládá ze dvou částí - serverové části (backend) a klientské části (frontend). Obě jsou realizovány v jazyce JavaScript. Jelikož má aplikace obě části psané ve stejném jazyce, je možno sdílet některá nastavení či balíčky. U větších aplikací komerčního použití je tato vlastnost často využívána ve velkém rozsahu.

Babel Babel je v obou částech aplikace použit pro kompilování javascriptu. Jelikož JavaScript prošel velkým vývojem v posledních letech, je třeba zajistit, aby byla aplikace spustitelná i na starších prohlížečích, které neobsahují implementace nejnovějších konstrukcí jazyka. Stejný problém je i u backendové části realizované pomocí NodeJS. U babelu byly využity presety stage-0 (testovací konstrukce jazyka) a ES2015. [5]

MongoDB MongoDB je dokumentová databáze. Dokumenty jsou uloženy ve formátu, který se podobá formátu JSON. MongoDB se stalo v posledních letech téměř standardem všech NodeJS aplikací. [21] Byla zvolena hlavně díky své flexibilitě. Také poskytuje velikou volnost v přístupu k agregacím a jednoduché rozhraní. Další výhodou jsou dynamické kolekce, známé z relačních databází jako tabulky.

4.1 Serverová část

Cílem práce nebylo programování serverové části aplikace, ale její klientské části. Avšak aplikace bez funkčního serverového API by nebyla použitelná. Většina JavaScriptových aplikací potřebuje API poskytující data. Zejména pak aplikace rozdělené na frontend a backend.

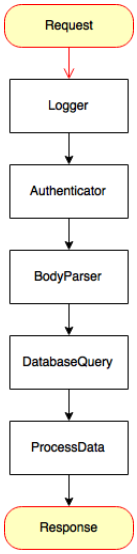
Jedním z dalších úkolů serverové části je zajištění bezpečnosti. Aplikace musí poskytovat pouze data, která jsou relevantní k přihlášenému uživateli. Klientská část je dostupná případným útočníkům. Je tedy nutno zajistit, aby všechny dotazy byly validovány na serveru.

Datové struktury poskytuje serverová část. Ta je implementována v jazyce JavaScript za pomocí frameworku Koa verze 2, dále jen Koa.

4.1.1 Koa

Koa je velice tenký backendový framework. Pomocí `async` funkcí jsou v Koa frameworku tvořeny pomocí middleware. Middleware je funkce sloužící k provádění operací v průběhu programu. Middleware je definována jako asynchronní funkce,

která obsluhuje dotaz. Ukázka middleware pro logování, autentizaci, zpracování obsahu requestu, dotazu na databázi a zpracování requestu (dotazu) do formy response (odpovědi) je znázorněna na obrázku 4.1.1. Middleware nejsou prováděny v pořadí, v jakém byly zadány, není li explicitně definována pomocí direktiv `async` a `await`.



Obr. 4.1: Ukázka requestu na Koa aplikaci

4.1.2 GraphQL

Backend slouží jako poskytovatel dat z pohledu vnějšího světa. Aby tato role byla zajištěna efektivně, byl použit dotazovací jazyk GraphQL od Facebooku. Ten umožňuje dotazovat se různá data pomocí jednoho dotazu.

U běžné Representational state transfer (REST) aplikace je normální se dotazovat na data následujícím způsobem. Aplikace chce zobrazit seznam úkolů. Seznam však zároveň zobrazuje název a adresu projektu. Proto, aby aplikace získala všechna potřebná data, je nutno se dotázat nejprve na seznam úkolů a poté na potřebné projekty.

```
{
  viewer {
    tasks {
      _id,
      number,
      name,
      done,
      created_at,
      project {
        name,
        url
      }
    }
  }
}
```

Ukázka 4.1: Query získání seznamu požadavků

GraphQL jazyk nabízí možnost získat všechna potřebná data efektivně na jeden dotaz. V dotazu je definován parametr `query`. `Query` obsahuje seznam informací, které potřebuje aplikace dostat v odpovědi. Backend pak dle zadaných definic získá data z uložiště a odešle je v odpovědi. Tím se velice efektivně pracuje se sítí a šetří cenný čas.

GraphQL umožňuje také pomocí takzvaných mutací provést vkládání, mazání nebo úpravy dat.

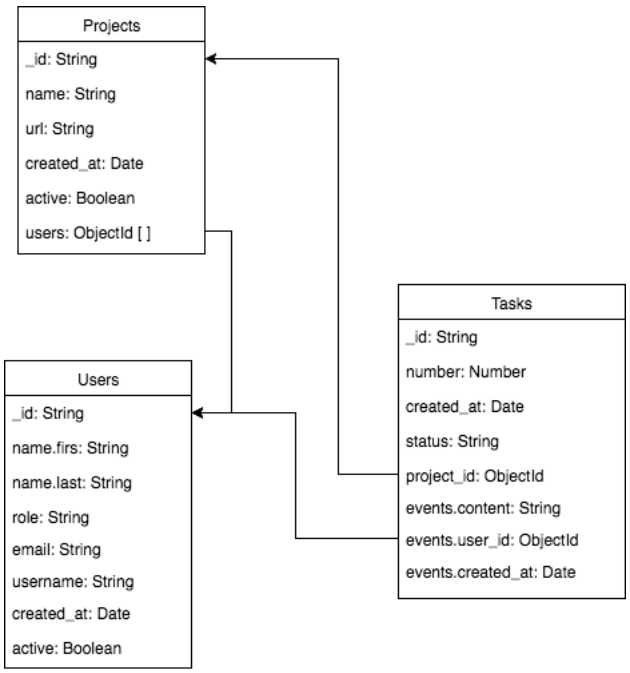
```
create(task: Task) {
  ...
  return this.graphQlClient.mutation(`
    mutation {
      taskCreate(input: {
        record: {
          name: "${task.name}",
          project_id: "${task.project_id}",
          events: [ ${eventsString} ]
        }
      }) {
        record{
          _id, number,
          name,
          created_at,
          done,
          events { user_id, content, created_at },
          project { name, url }
        }
      }
    }
  `).then(result => {
    if (typeof result.errors !== 'undefined') {
      return result;
    }

    const record = result.data.taskCreate.record;
    ...
    return task;
  }).catch(err => err);
}
```

Ukázka 4.2: Mutace vložení nového úkolu

4.1.3 Mongoose

Modelová vrstva z MVC je realizována pomocí mongoose. Mongoose je prostředník mezi GraphQL a MongoDB. Mongoose poskytuje resolvery (řešitele) pro získávání dat z MongoDB. Pomocí jeho mapování entit jsou pak definovány i entity pro GraphQL.

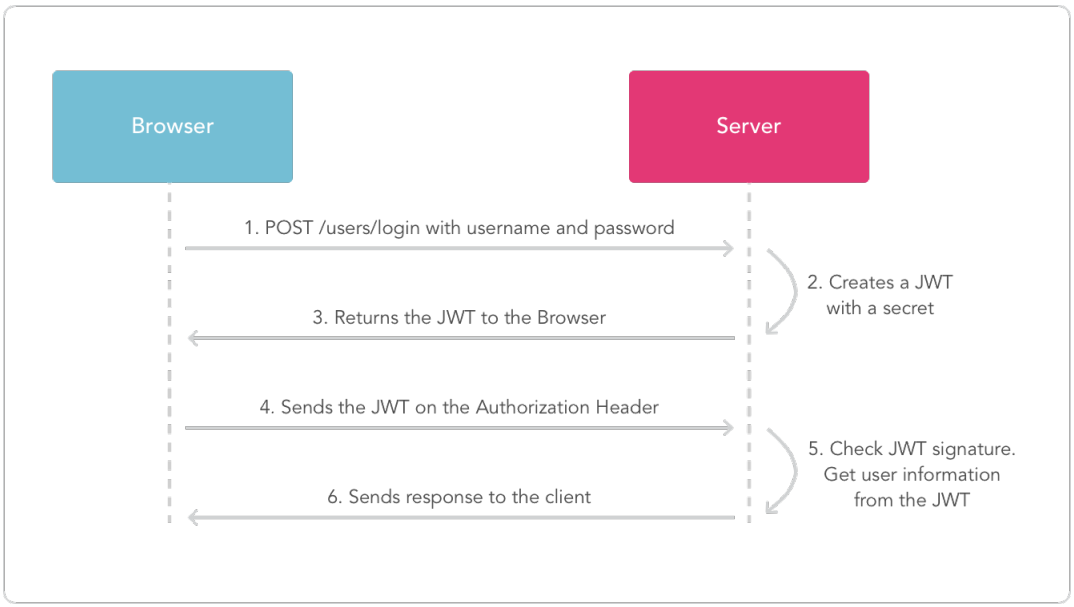
Mongoose je vybrán díky jeho komplexnosti a oblíbenosti v komunitě. Staví na něm projekty jako LinkedIn, Trello nebo Storify. Jeho definice entit jsou velice jednoduché, rychlé a přehledné. Zároveň poskytuje API pro dotazování dat se všemi potřebnými možnostmi.



Obr. 4.2: Struktura aplikačního modelu

4.1.4 JWT

JWT je otevřený standart RFC 7519 [20], sloužící k reprezentaci bezpečné komunikace mezi dvěma stranami. JWT je využit k autentizaci uživatele. Na obrázku 4.1.4 je znázorněno jeho použití. Backend při autentizaci uživatele vygeneruje podepsaný JWT token. Ten je zaslán v odpovědi klientské části. Klientská část jej pak zasílá při každém dotazu na backend v hlavičce `Authorization`. Backend tuto hlavičku používá, aby ověřil přihlášeného uživatele a jeho objekt přiřadil do kontextu. Později je pomocí tohoto kontextu rozhodnuto, zda má uživatel práva k provedení dotazované operace.



Obr. 4.3: Schéma autorizace pomocí JWT tokenu [4]

4.1.5 PassportJS

PassportJS je autentizační middleware poskytující rozhraní pro autentizaci uživatelů. Jeho výhodou je možnost implementace autentizace uživatelů za minimálního programování. Výsledkem implementace v PassportJS jsou dvě funkce. Jedna obsluhující registraci a druhá generující JWT token.

V ukázce 4.3 je znázorněna konfigurace základních částí PassportJS a funkce pro generování JWT tokenu. Konfigurace musí obsahovat povinné části serializace a deserializace uživatele. Dále je nutno definovat funkci, která dle získaného payloadu z requestu dokáže vyhledat uživatele.

```
1  import passport from 'koa-passport';
2  import {Strategy as JWTStrategy, ExtractJwt} from 'passport-jwt';
3  import jwt from 'jsonwebtoken';
4  import config from 'config';
5
6  import UserModel from '../models/users';
7
8  passport.use(UserModel.createStrategy());
9
10 // use static serialize and deserialize of model for passport
       session support
11 passport.serializeUser(UserModel.serializeUser());
12 passport.deserializeUser(UserModel.deserializeUser());
13
14 const opts = {
```

```
15    jwtFromRequest: ExtractJwt.fromAuthHeader(),
16    secretOrKey: config.get('secret'),
17  };
18
19  // auth user by payload
20  passport.use(new JWTStrategy(opts, async(jwt_payload, done) => {
21    const user = await UserModel.findOne({_id: jwt_payload.id});
22    if (user) {
23      done(null, user);
24    } else {
25      done(null, false);
26    }
27  }));
28
29  // After autentication using one of the strategies, generate a JWT
      token
30  export function generateToken() {
31    return async ctx => {
32      const {user} = ctx.state;
33      if (!user) {
34        ctx.status = 401;
35      } else {
36        const _token = jwt.sign({
37          id: user.id,
38          role: user.role
39        }, config.get('secret'));
40        const token = `JWT ${_token}`;
41
42        ctx.status = 200;
43        ctx.body = {
44          token
45        };
46      }
47    };
48  }
```

Ukázka 4.3: Konfigurace PassportJS pro identifikaci uživatele v kontextu

Aby bylo možno uživatele přihlásit nebo registrovat, je nutno definovat příslušné routy. Snadnější definicí je routa pro přihlášení. PassportJS nabízí již interně implementovanou funkci, která dokáže vyhledat uživatele a autorizovat jej dle přihlašovacího jména a hesla. Registrace uživatele je třeba provádět ručně. Jak je vidět v ukázce 4.4, registrace je však také velice jednoduchá. Je pouze třeba za pomocí direktiv `async/await` vyčkat na vyřešení Promise, která obaluje samotnou funkci registrace implementovanou přímo v modelu.

```
import Router from 'koa-router';
import passport from 'koa-passport';

import {generateToken} from './';

import UserModel from '../models/users';

let router = new Router;

router.post('/login', passport.authenticate('local'), generateToken
    ());
router.post('/register', async(ctx, next) => {
  let user = new UserModel(ctx.request.body);

  return await new Promise(function (resolve, reject) {
    UserModel.register(user, ctx.request.body.password, async(err,
        user) => {
      if (err) {
        reject(err);
      }

      ctx.state.user = user.toObject();
      return resolve(next());
    });
  });
}, generateToken());

export default router;
```

Ukázka 4.4: Autentizace a registrace v PassportJS s využitím JWT

4.2 Klientská část

Klientská část je hlavním předmětem práce. Jako realizační framework byla vybrána Aurelia. Hlavním důvodem byla nezmapovanost frameworku a jeho potenciál. Během realizace bylo odhaleno několik slabých míst, ale také několik velice povedených implementací.

4.2.1 Aurelia

Aurelia je značně pohodlný framework, který po celou dobu byl kvalitním průvodcem realizací aplikace. Nejnepříjemnějším problémem je dostupnost zdrojů a ukázek. Jelikož se jedná o méně rozšířený framework, neexistuje tolik manuálů, návodů ani projektů k inspiraci či porovnání.

Naopak nejpříjemnější částí jsou definice tříd kontrolérů. Ty jsou zapsány pouze v nové ES2016 syntaxi. Není tedy třeba příliš přemýšlet o tom, jak třídu napsat. Jejich tvorba je velice intuitivní podobná třídám z jiných jazyků, jako je například Java, PHP nebo dalších OOP jazyků. Mezi nedostatky by se měl také zařadit návrh kontrolérů. V JavaScriptu je velice běžné nechávat příliš mnoho funkčnosti na kontrolérech. Ty by dle MVC měly obstarávat pouze dotaz, v případě JavaSctipu ještě události jako kliknutí. Dalším problémem je validace dat. Ať už se jedná o validaci jako takovou, nebo validaci formulářů. Aurelia nepodporuje validaci zanořených objektů. Tedy pokud je třeba validovat úkol a jeho vlastnost projekt, je třeba definovat si vlastní validaci.

Každá jednotlivá stránka v aplikaci má svůj vlastní kontroler. Jeho zápis v Aurelii je velice jednoduchý a srozumitelný. Ukázka 4.5 znázorňuje kontroler přihlašovací stránky aplikace.

```
1  import {inject, NewInstance} from 'aurelia-framework';
2  import {ValidationController} from 'aurelia-validation';
3  import {AuthService} from 'aurelia-auth';
4
5  @inject(NewInstance.of(ValidationController), AuthService)
6  export class Login {
7    controller: ValidationController = null;
8
9    constructor(controller: ValidationController, authService:
       AuthService) {
10     this.controller = controller;
11     this.authService = authService;
12
13     this.username = 'test';
14     this.password = 'test';
15   }
16
17   submit() {
18     return this.authService.login({username: this.username,
         password: this.password})
19       .then(response => {
20         console.log("success logged " + response);
21       })
22       .catch(err => {
23         this.controller.errors = []; // clear
24         this.controller.addError('Login failed');
25       });
26   };
27
28   authenticate(name) {
29     return this.authService.authenticate(name, false, null)
```

```
30        .then((response) => {
31          console.log("auth response " + response);
32        });
33    }
34  }
```

Ukázka 4.5: Kontroler přihlašovací stránky

4.2.2 Aurelia CLI

Pro spuštění aplikace je využíváno Aurelia CLI balíčku. Ten se řídí konfigurací uloženou v souboru `aurelia_project/aurelia.json`. Soubor `aurelia.json` obsahuje konfiguraci transpileru, CSS procesoru, cest ke statickým datům nebo také závislosti vyžadované ke spuštění. [2]

Aurelia využívá Gulp ke spouštění úloh, jako je transpilace JavaScriptu pomocí babelu, minifakaci kódů a dalších. Gulp je popsán v kapitole 1.4.3. Jedná se o nástroj umožňující spouštět úlohy, které zajišťují mnohem větší programátorský komfort.

Dalším použitým balíčkem je browser sync. Ten zajišťuje automatické přenačtení stránky v případě uložení změn ve zdrojových kódech. Browser sync je však použit pouze v případě, že je aplikace spuštěna s příznakem `-watch`. Hlavní výhodou automatického přenačtení stránky je, že při vývoji webových aplikací je žádoucí, aby při každém uložení změn vývojář obnovil webovou stránku. Automatizací tohoto procesu lze docílit značné časové úspory.

4.2.3 Další knihovny a utility

K realizaci byly použity také další knihovny či utility, jedná se o menší části hotových nástrojů. Ty usnadňují realizaci některých částí aplikace.

Minimalistickou utilitou, usnadňující práci s tabulkami dat, je Aurelia table. Ta byla použita při realizaci všech tabulek v aplikaci. Hlavní výhodou utility je možnost jednoduchých a rychlých filtrací. Pouze za pomocí definice dvou proměnných a jedné funkce lze dosáhnout komplexního řešení. [24] Ukázka 4.6 demonstruje konfiguraci filtru pro vyhledávání v úkolech. Ukázka zobrazuje vyhledávání za využití vlastností: název úkolu, název projektu, URL adresa projektu a čísla úkolu.

```
1  ...
2  export class Tasks {
3    tasks = [];
4
5    filters = [
6      {value: '', keys: ['name', 'project.name', 'project.url', '
           number']},
7    ];
```

```
8
9     ...
10 }
```

Ukázka 4.6: Definice filtru v Aurelia table

Pro obsluhu promisí Aurelia využívá knihovnu Bluebird. Jelikož celá aplikace je asynchronní, je třeba zajistit synchronicitu dat. Standardní implementace JavaScriptových promisí není pro Aurelii dostatečná.

MomentJS je použit pro snadnější manipulaci s daty. Jeho využití je minimalistické. Slouží jako převodník hodnot `date-format` k formátování objektu s datem do řetězce čitelného v časovém pásmu České republiky.

```
1  import moment from 'moment';
2
3  export class DateFormatValueConverter {
4    toView(value, format) {
5      return moment(value).format(typeof format !== 'undefined' ?
           format : 'M.D.YYYY h:mm:ss');
6    }
7  }
8  }
```

Ukázka 4.7: Využití MomentJS jako převodníku hodnot

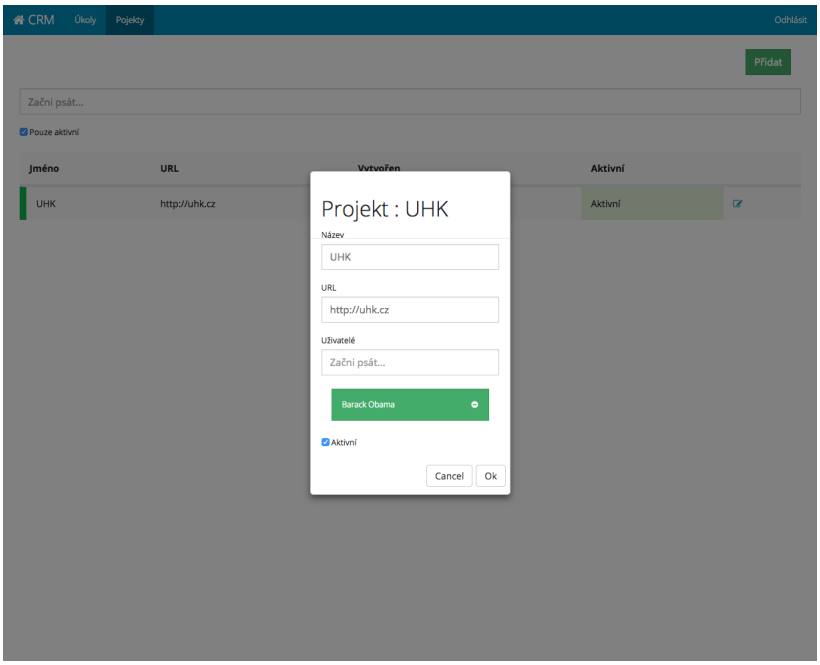
4.2.4 Uživatelské rozhraní

Uživatelské rozhraní je realizováno v CSS frameworku bootstrap. Níže jsou obsaženy snímky obrazovky ukazující vybrané části aplikace. Bootstrap je CSS a JavaScriptová knihovna obsahující hotové komponenty. Příkladem mohou být formuláře, menu či layout. [23] Font-Awesome slouží k zobrazování ikon. Jeho využití je zajímavé v ohledu minimální investice času do obohacení odkazů ikonami. Utilita Font-Awesome obsahuje několik velikostí a je kompatibilní s většinou prohlížečů. [14] NProgress zobrazuje status bar načítání se stránky. Jeho úlohou je usnadnit uživateli interakci s aplikací. [25]

Jelikož je využit moderní CSS framework bootstrap, aplikace nabízí také responzivní zobrazení. V ukázce 4.2.4 je zobrazení přihlašovací obrazovky na tabletu s rozlišením 768x1024px.

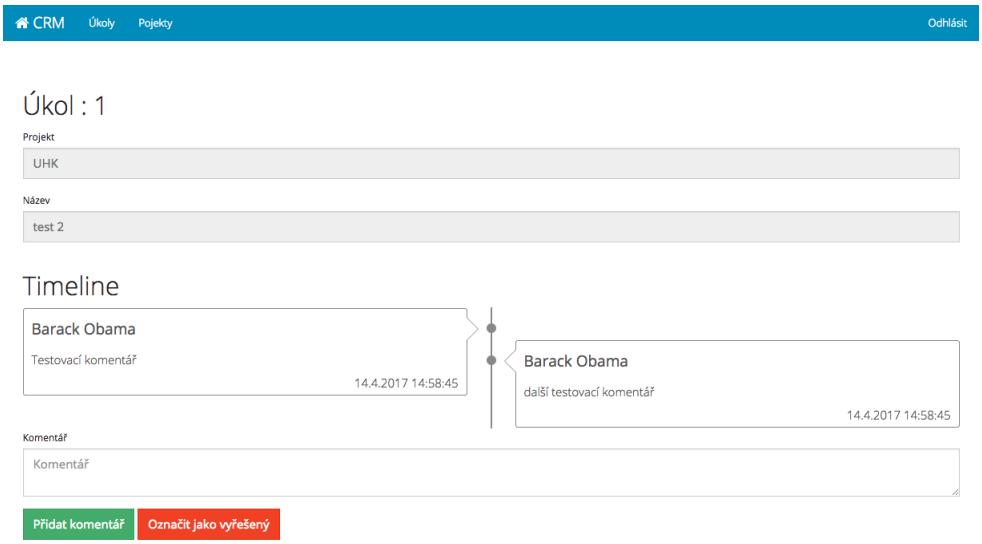
Obr. 4.4: Přihlašovací obrazovka na tabletu

Z důvodu demonstrace různých přístupů k manipulaci s entitou jsou v aplikaci použity dva typy editace. Prvním je editace v dialogovém okně zobrazena na obrázku 4.2.4. Tento dialogový přístup umožňuje editovat projekt bez nutnosti přesměrování stránky a opětovného načítání dat ze strany backendu. Jeho výhodou jsou tedy úspora síťových prostředků a vyšší interaktivnost. Další možností by mohlo být využití velkého rozlišení, které nabízí dnešní počítače, a zobrazit editaci na pravé straně obsahu v panelu. To by však snížilo možnosti responzivnosti (přizpůsobení se šířce).



Obr. 4.5: Editace projektu

Další možností je vytvoření samostatné editační stránky, jak tomu bylo zvykem zejména v minulosti. Tento přístup umožňuje zobrazení většího množství prvků. Hodí se zejména na místa, kde je třeba zadávat nebo zobrazovat větší počet prvků. Příkladem je detail úkolu 4.2.4. V detailu úkolu je třeba ještě k samotným vlastnostem přidat také časovou osu komentářů. Ta by v dialogovém okně nepůsobila příliš přehledně.



Obr. 4.6: Detail úkolu

4.2.5 Zabezpečení

Jednou z nejproblematičtějších částí JavaScriptové aplikace je bezpečnost. V případě ukázkové aplikace jsou využity JWT, kdy backend zašle po úspěšné autentizaci JWT v odpovědi. Klientská část jej pak používá při každém dotazu v hlavičce `Autrorization` dotazu. Toto řešení by v případě komerční aplikace nebylo dostatečné.

K autorizaci je využito technologie JWT popsané v kapitole 4.1.4. Implementace JWT je v klientské části realizována pouze pomocí několika řádku nastavení viz. 4.8. Využívá se totiž balíčku `aurelia-auth`. Ten se stará o veškerou funkčnost autentizace a autorizace.

```
1 const configForDevelopment = {
2   // Our Node API is being served from localhost:3001
3   baseUrl: 'http://localhost:3000',
4   // The API specifies that new users register at the POST /users
      enpoint.
```

```
5     signupUrl: 'auth/register',
6     // Logins happen at the POST /sessions/create endpoint.
7     loginUrl: 'auth/login',
8     loginRedirect: '#/tasks',
9
10    responseTokenProp: 'token',
11    authToken: '',
12    providers: {
13    }
14  };
15
16  let config;
17  if (window.location.hostname === 'localhost') {
18    config = configForDevelopment;
19  }
20  else {
21   config = configForProduction;
22  }
23
24  export default config;
```

Ukázka 4.8: Aurelia-auth konfigurace

5 VÝSLEDKY A ZÁVĚR

Práce je zaměřena na srovnání JavaScriptových frameworků. Porovnání je velmi aktuální téma, avšak v čase zastarává. Pomocí zvoleného frameworku Aurelia byla implementována testovací aplikace. Na ní byly demonstrovány konstrukce dle frameworku. Aurelia je kvalitním nástrojem pro vývoj moderní a dynamické aplikace v jazyce JavaScript.

Výsledkem práce je srovnání JavaScriptových frameworků se zaměřením na komunitu, velikost v kB, na šablonovací systém a základní konstrukce kontrolérů. Poslední dvě kapitoly jsou zaměřeny na implementaci jednoduché CRM aplikace. Ta obsahuje několik uživatelských rolí s rozdílnými oprávněními, možnost zakládat úkoly a projekty. Klienti mají práva manipulace s úkoly. Vývojáři rozšiřují oprávnění klientů o manipulaci projektů, správci pak o možnost správy uživatelů.

Hlavním úkolem aplikace je demonstrace vývoje ve zvoleném frameworku. Aplikace využívá jazyku JavaScript a databázového systému MongoDB. Byla navržena tak, aby splňovala požadavky moderní doby a zároveň byla jednoduše rozšířitelná o novou funkčnost. Hlavním přínosem práce je zmapování a přehled současných trendů v oblasti frontendových frameworků. Důraz byl kladen na srozumitelnost porovnání, a to pomocí praktických ukázek, aby po přečtení bylo možno zvolit framework, který nejlépe odpovídá požadavkům aplikace. Dalším z přínosů práce je zmapování frameworků s menšími komunitami jako v případě Aurelie.

V současnosti je k dispozici vysoké množství balíčků, což naznačuje, že v budoucnosti bude JavaScript zahlcen velkým množstvím neudržovaných knihoven. Je možné očekávat, že velké frameworky jako Angular budou i v budoucnosti podporovány větším množstvím vývojářů. Každý z frameworků používá rozdílné přístupy, nedá se tedy očekávat, že by proběhla fúze mezi některými z nich.

LITERATURA

[1] Angular community. angular/angular [online]. Dostupné z URL: https://github.com/angular/angular, 22. 3. 2017.

[2] Aurelia community. aurelia/cli [online]. Dostupné z URL: https://github.com/aurelia/cli, 27. 3. 2017.

[3] Aurelia community. aurelia/framework [online]. Dostupné z URL: https://github.com/aurelia/framework, 25. 3. 2017.

[4] Auth0. Jwt introduction [online]. Dostupné z URL: https://jwt.io/introduction/, 8. 4. 2017.

[5] Babel. Babel · the compiler for writing next generation javascript [online]. Dostupné z URL: http://babeljs.io/, 13.4. 2017.

[6] Backbone community. backbone.js [online]. Dostupné z URL: http://backbonejs.org, 15. 1. 2017.

[7] Backbone community. jashkenas/backbone [online]. Dostupné z URL: https://github.com/jashkenas/backbone, 26. 3. 2017.

[8] Ecma International. Ecma-262 [online]. Dostupné z URL: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf, 3. 1. 2017.

[9] Ecma International. Ecmascript® 2015 language specification [online]. Dostupné z URL: http://www.ecma-international.org/ecma-262/6.0/#sec-arrow-function-definitions, 22. 1. 2017.

[10] Ember community. emberjs/ember.js [online]. Dostupné z URL: https://github.com/emberjs/ember.js, 30. 3. 2017.

[11] Facebook Inc. React native [online]. Dostupné z URL: https://facebook.github.io/react-native/, 5. 12. 2016.

[12] Facebook inc. Yarn [online]. Dostupné z URL: https://yarnpkg.com/, 18. 1. 2017.

[13] Flanagan, D. *JavaScript: The Definitive Guide*. O'Reilly Media, Inc., 2006. 1096 stran.

[14] Gandy, D. Font awesome, the iconic font and css toolkit [online]. Dostupné z URL: http://fontawesome.io/, 10. 4. 2017.

[15] Github Inc. Electron [online]. Dostupné z URL: http://electron.atom.io/, 12. 12. 2016.

[16] Google inc. Google trends [online]. Dostupné z URL: https://www.google.com/trends/explore?date=2010-01-01%202016-12-31&q=emberjs,angularjs,backbonejs,aureliajs&hl=en-US, 12. 1. 2017.

[17] Grunt community. Sample gruntfile [online]. http://gruntjs.com/sample-gruntfile, 24. 2. 2017.

[18] Gulp community. gulpjs.com [online]. Dostupné z URL: http://gulpjs.com, 3. 3. 2017.

[19] Io.js comunity. Roudmap io.js [online]. Dostupné z URL: http://roadmap.iojs.org, 1. 1. 2017.

[20] Jones, M., Microsoft a ostatní. RFC 7519. Dostupné z URL: https://tools.ietf.org/html/rfc7519, 15. 1. 2017.

[21] Mongo DB Inc. What is mongodb? | mongodb [online]. Dostupné z URL: https://www.mongodb.com/what-is-mongodb, 10. 4. 2017.

[22] Node.js foundation. Node.js [online]. Dostupné z URL: https://nodejs.org/en/, 10. 3. 2016.

[23] Otto, M., Thornton, J., and přispivatelé, B. Bootstrap - the world's most popular mobile-first and responsive front-end framework. [online]. Dostupné z URL: http://getbootstrap.com/, 3. 3. 2017.

[24] Romero, H. Aurelia table [online]. Dostupné z URL: https://tochoromero.github.io/aurelia-table/, 4. 3. 2017.

[25] Sta, R. Nprogress: slim progress bars in javascript [online]. Dostupné z URL: http://ricostacruz.com/nprogress/, 7. 3. 2017.

[26] Teixeira, P. *Hands-on Node.js [ebook].* Leanpub, Dostuné z URL: https://leanpub.com/hands-on-nodejs, 2016.

[27] Škultéty, R. *JavaScript: kapesní přehled.* Compiter Press, a.s., 2005.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

UX	user experience
OOP	objektově orientované programování
CLI	client line interface
HTML	HyperText Markup Language
CSS	Cascading Style Shets
CSS	Cascading Style Sheets
SASS	Syntactically Awesome Style Sheets
ES5	ECMAScript 5th edition
ES6	ECMAScript 6th edition
ES2015	ECMAScript 2015
ES2016	ECMAScript 2016
ES2017	ECMAScript 2017
MVC	Model View Controller
MVP	Model View Presenter
DI	dependency injection
IDE	Integrated development environment
RC	Release Candidate
BC	Breaking-change
CRM	Customer Relationship Management
API	Application Programming Interface
JWT	JSON Web Tokens
REST	Representational state transfer

SEZNAM PŘÍLOH

A JSON PRO VYTVOŘENÍ PRVNÍHO UŽIVATELE

```
{
  "role": "admin",
  "email": "emai@vhrb.cz",
  "username": "test",
  "active": true,
  "name": {
    "first": "Test",
    "last": "Test"
  },
  "password": "test"
}
```

Ukázka A.1: JSON pro vytvoření prvního uživatele

B DATOVÉ MÉDIUM

B.1 Popis struktury

- složka `crm-fe` obsahuje soubory aplikace zajišťující běh uživatelské části,
  - `aurelia_project` je seznam všech nastavení a úkolů pro aurelia-cli,
    - `enviroments` obsahuje soubory s konfiguracemi pro jednotlivá prostředí od vývojového po produkční,
    - `generators` obsahuje generátory komponent Aurelie,
    - `tasks` obsahuje úlohy pro kompilaci a transpilaci,
    - `aurelia.json` obsahuje nastavení pro aurelia-cli,
  - `src` obsahuje zdrojové kódy aplikace,
    - `components` je složka s komponentami všech stránek aplikace. Obsahuje HTML soubory i kontrolery,
    - `css` obsahuje všechny CSS soubory,
    - `entity` je složka se všemi datovými schránkami,
    - `resources` složka obsahující jednoduché komponenty jako převodníky hodnot,
    - `services` obsahuje všechny služby, které jsou použity pro načítání a ukládání dat,
    - `acl-step.js` je validátor přístupů na stránku,
    - `app.html` je základní HTML layout,
    - `app.js` obaluje třídu obsahující seznam stránek a poskytující obsluhu authentizaci,
    - `authConfig.js` obsahuje konfiguraci pro balíček `aurelia-auth`,
    - `enviroment.js` je generovaný soubor Aurelii dle spuštěného enviromentu,
    - `main.js` spuští aplikaci a připojuje do ní balíčky,
    - `message.js` je seznam obsluhovaných zpráv pro `EventAggregator`,
  - `.babelrc` je konfigurace pro balíček babel,
  - `index.hml` je hlavní soubor HTML,
  - `package.json` obsahuje seznam balíčků, které jsou nezbytné pro běh aplikace,
- složka `crm-be` obsahuje soubory aplikace pro backend,
  - `config` obsahuje základní konfiguraci MongoDB a serveru,
  - `src` jsou zdrojové soubory backendu,
    - `auth` obsahuje soubory pro obsluhu authentizace,
    - `graphql` obsahuje sestavení graphql schématu,

    - `initializator` je složka s inicializačními soubory pro MongoDB, session a server,
    - `models` obsahuje modely entit pro Mongoose,
    - `router` obsahuje soubor pro sestavení rout v potřebném pořadí,
    - `index.js` je hlavním souborem aplikace postupně přidávajícím middleware do aplikace,
  - `.babelrc` je konfigurace pro balíček babel,
  - `package.json` obsahuje seznam balíčků, které jsou nezbytné pro běh aplikace,

B.2 Návod ke spuštění pod UNIX platformou

1. Nainstalujeme databázi MongoDB. Dle platformy a návodu uvedeného na adrese: `https://docs.mongodb.com/manual/installation/`.
2. Nainstalujeme balíčkovací systém `npm`. Dle platformy uvedené na adrese: `https://nodejs.org/en/download/`.
3. Nakopírujeme složky `crm-be` a `crm-fe` do cílových složek na serveru nebo lokálním počítači.
4. V terminálu přejdeme do složky `crm-be`.
5. Spustíme příkaz `npm install`.
6. Po instalaci pustíme příkazem `npm start`. Backend se spustí na adrese `http://localhost:3000/`.
7. V novém okně terminálu přejdeme do složky `crm-fe`.
8. Spustíme příkaz `npm install`.
9. Pomocí příkazu `au run` spustíme aplikaci uživatelského rozhraní. Uživatelské rozhraní běží pod adresou `http://localhost:9000/`.
10. Nyní je třeba nastavit administrátorem preferovaný server. Ten slouží jako proxy na uživatelské rozhraní. Tedy je třeba pomocí jej směřovat na `localhost:9000`.
11. Před prvním přihlášením je třeba vytvořit administrátora. Provedením POST (viz. příloha ‘A.1) requestu na backend ve formátu JSON.
12. Přihlásíme se do uživatelského rozhraní pomocí uživatele "test" s heslem "test".

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2016/2017

Studijní program: Aplikovaná informatika
Forma: Kombinovaná
Obor/komb.: Aplikovaná informatika (ai3-k)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Dobrovolný Michal	Strachotice 138, Strachotice	I14488

TÉMA ČESKY:

Moderní javascriptové frameworky

TÉMA ANGLICKY:

Modern javascript frameworks

VEDOUCÍ PRÁCE:

doc. Mgr. Tomáš Kozel, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl: Popis aktuálních javascriptových frameworků a vytvoření vzorové aplikace ve vybraném frameworku.

Osnova:
1. Úvod
2. Javascript a jeho historie
3. Srovnání frameworků
4. Analýza a návrh aplikace
5. Popis implementace
6. Výsledky a závěr

SEZNAM DOPORUČENÉ LITERATURY:

-

Podpis studenta: ..... **Datum:** .....

Podpis vedoucího práce: ..... **Datum:** .....